PROGRAM CREATION METHOD AND PROGRAM EXECUTION METHOD

[0001]

FIELD OF THE INVENTION

The present invention relates to a program creation method and a program execution method, and more particularly to a technology for preventing software from being analyzed dynamically.

[0002]

BACKGROUND OF THE INVENTION

Conventionally, a software illegal-use prevention technology has been used to prevent a third party from analyzing a high-security program and illegally using it, as described, for example, in "Protection of software against reverse-analysis (tampering) and alteration (modification) - tamper-resistant software technology -", pp. 209-220, Nikkei Electronics Vol. No. 706, January, 1998".

[0003]

One of methods for illegally using software is to use a software debugger etc. to get access to a program in execution for inspecting the behavior of the program by executing the program statement by statement. This is a powerful analysis means for which no complete prevention method is available.

[0004]

The following describes a dynamic-analysis prevention technology, as one of illegal-use prevention technologies, described in the document mentioned above. FIG. 8 shows the configuration of the technology. Referring to FIG. 8, programs 10 and 20 are programs to be protected by using the illegal-use prevention technology, and a program 60 is a supervisory program, (System Integrity Program SIP) prepared for the illegal-use prevention technology. Note that the programs 10, 20, and 60 each include an alteration detection code module (Integrity Verification Kernel IVK) 32 for detecting alterations that might be effected on the program itself, respectively.

[0005]

These programs perform authentication (verification) as follows. The program 10 and the program 60 authenticate (verify) the alteration detection code module (IVK) 32 each other according to a communication protocol through digital signature. In this case, if the alteration detection code module 32 is destroyed or altered, processing stops here immediately.

[0006]

The program 20 and the program 60 authenticate the alteration detection code module 33 each other according to a communication protocol through digital signature. In this case, if the alteration detection code module 32 is destroyed or altered, processing stops here immediately.

[0007]

This method allows the processing to be caused/suspended, if the alteration detection code module 32 detects that a dynamic analysis was made. In addition, this method performs authentication for two pairs of programs to prevent an illegal use by a third party program that imitates a communication protocol message.

[0008]

SUMMARY OF THE DISCLOSURE

One of the problems with the software illegal-use prevention technology described above resides in that, because the alteration detection code module detects merely alterations, the technology can not detect tracing itself underway performed by a software debugger but detects alterations only when the debugger has altered some part of the program.

[0009]

In view of the foregoing, it is an object of the present invention to provide a program creation method and a program execution method that solve the above problem and prevent a software debugger from making a dynamic software analysis.

[0010]

According to a first aspect of the present invention there is provided a program creation method wherein encoded code modules are made up of first and second programs that decrypt encoded code modules each other during execution, each of the encoded code modules being generated by encrypting corresponding one of the processing code modules.'

[0011]

According to a second aspect of the present invention there is provided a program execution method wherein, during execution of first and second programs, encrypted code modules are decrypted each other, each of the encoded code modules being generated by encrypting one of processing code modules corresponding to the first program and the second program, respectively.

[0012]

That is, in the program creation method of the present invention, two programs each having encrypted code modules decrypt the encrypted code modules each other during execution to prevent a software debugger from dynamically analyzing the programs.

[0013]

More specifically, in the program creation method of the present invention, the first program includes encrypted code modules to be executed at an odd-numbered occurrence during whole processing operation and the second program includes encrypted code modules to be executed at an even-numbered occurrence during the whole processing operation. The first and second programs are configured so that they execute encrypted code modules each other alternately. Any encrypted code module cannot be returned (decrypted) to the original code module unless being decrypted by both the first and second programs.

[0014]

When the first and second programs are started, the first encrypted code module is decrypted and the first program executes the resultant decrypted code module. Next, the first and second programs decrypt the second encrypted code module and the second program executes the resultant decrypted code module.

[0015]

By repeating the above processing, the encrypted code module cannot be decrypted completely, when a software debugger attempts to dynamically analyse the first program. The second program, which attempts to execute an incompletely decrypted code module, is terminated because it executes an unauthorised or improper code. The subsequent encrypted code modules are not decrypted completely and, therefore, the first program is also terminated.

[0016]

The present invention takes advantage of the fact that a software debugger can start and analyze one program at a time and the fact that it takes a longer time to execute dynamic analysis than to execute a standard program. These facts allow programs to be structured to prevent illegal dynamic analysis.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram showing the structure of programs used in an embodiment of the present invention.

FIG. 2 is a diagram showing programs before being encrypted as in FIG. 1.

FIG. 3 is a diagram showing a sequence in which the programs in FIG. 1 are executed.

FIG. 4 is a diagram showing the programs that are operating correctly.

FIG. 5 is a flowchart showing the operation of the programs in FIG. 1.

FIG. 6 is a diagram showing the programs in FIG. 1 that are being dynamically debugged.

FIG. 7 is a diagram showing the structure of programs in another embodiment of the present invention.

FIG. 8 is a diagram showing a conventional dynamic analysis prevention method.

[0017]

PREFERRED EMBODIMENTS OF THE INVENTION

Some embodiments of the present invention will now be described with reference to the drawings. FIG. 1 is a diagram showing the configuration of programs used in one embodiment of the present invention. Referring to FIG. 1, a (first) program 10 comprises encrypted code modules (i. e., encrypted processing code modules) 11, 13, and 15, decryption code modules 12a, 14a, and 16a of encrypted code modules 12, 14, and 16, and a decryption processing code module 30. Also, a (second) program 20 comprises the encrypted code modules 12, 14, and 16,' decryption code modules 11a, 13a, and 15a of the encrypted code modules 11, 13, and 15, and a decryption processing code module 31.

[0018]

FIG. 2 is a diagram showing the programs in FIG. 1 before being encrypted. In Fig. 2, processing code modules 1, 3, 5 (i.e., add-numbered) and the first decryption processing code 30 are contained in the first program 10, whereas processing code modules 2, 4 and 6 (i.e., add-numbered) and the second decryption processing code 31 are contained in the second program 20. FIG. 3 shows the encrypted code modules 11 to 16 generated by encrypting processing code modules 1-6, respectively. To return those encrypted modules to the processing code modules 1-6 that perform original processing, the encrypted code modules must be decrypted by the decryption processing code module 30 and/or the decryption processing code module 31.

[0019]

In one embodiment, the decryption processing code module 30 decrypts the encrypted code modules 12, 14, and 16 of the program 20 with the decryption code modules 12a, 14a, and 16a, respectively. Similarly, the decryption processing code module 31 decrypts the encrypted code module 11, 13, and 15 of the program 10 with the decryption code modules 11a, 13a, and 15a, respectively.

[0020]

FIG. 3 shows an operation sequence in which the programs shown in FIG. 1 are executed. Referring to FIG. 3, the programs 10 and 20 execute the encrypted code modules 11, 12, 13, 14, 15, and 16 in a time-serial manner.

[0021]

There are many ways of creating encrypted code modules to be executed time-serially in separate programs. In one way, the programs are created using the synchronization mechanism such as the system timer; in another way, encrypted code modules are used for processing modules that are executed in the fixed order such as initialization processing modules, screen drawing (displaying) processing modules, termination processing modules, and so on. It should be noted that the modules must be created such that processing will not be stopped through analysis by a software debugger. When creating encrypted code modules, it is desirable that a system timer be used as a waiting mechanism.

[0022]

FIG. 4 is a diagram showing the programs in FIG. 1 that are operating correctly. FIG. 5 is a flowchart showing the operation of the programs in FIG. 1. With reference to FIG. 1, FIG. 4, and FIG. 5, the overall operation of the embodiment of the present invention will be described.

[0023]

First, when the first program 10 and the second program 20 are started, the first decryption processing code module 30 and the second decryption processing code module 31, which uses the decryption code module 11a, work together to decrypt the encrypted code module 11 to generate the processing code module 1 (steps S1 and S11 in FIG. 5).

[0024]

At this time, the program (here, first program 10) waiting for decryption uses the waiting mechanism (step S2 in FIG. 5). This waiting mechanism is a routine to synchronize two concurrently running programs, first program 10 and second program 20. The waiting mechanism S2 waits a pre-calculated period of time until the second program 20 decrypts the encrypted code module 11. When the calculated period of time has elapsed, the next step is executed regardless of whether or not the encoded code is decrypted.

[0025]

After that, the first program 10 executes the processing code module 1 (step S3 in FIG. 5). Then, the first decryption processing code module 30, which uses the decryption code module 12a, and the second decryption processing code module 31 work together to decrypt the encrypted code module 12 to generate the processing code module 2 (steps S4 and S12 in FIG. 5).

[0026]

At this time, the (second) program waiting for decryption uses the waiting mechanism (step S13 in FIG. 5). After that, the second program 20 executes the processing code module 2 (step S14 in FIG. 5). Then, the first decryption processing code module 30 and the second decryption processing code module 31, which uses the decryption code module 13a, work together to decrypt the encrypted code module 13 to generate the processing code module 3 (steps S5 and S15 in FIG. 5).

[0027]

At this time, the (first) program waiting for decryption uses the waiting mechanism (step S6 in FIG. 5). After that, the first program 10 executes the processing code module 3 (step S7 in FIG. 5). Then, the decryption processing code module 30, which uses the decryption code module 14a, and the decryption processing code module 31 work together to decrypt the encrypted code module 14 to generate the processing code module 4 (steps S8 and S16 in FIG. 5).

[0028]

At this time, the second program waiting for decryption uses the waiting mechanism (step S17 in FIG. 5). After that, the second program 20 executes the processing code module 4. The first and second programs 10 and 20 continue processing in this manner.

[0029]

This procedure prevents the programs 10 and 20 from being analyzed even when a debugger (not shown in the figure) attempts to dynamically debug either one of the programs 10 and 20.

[0030]

FIG. 6 shows a state where a debugger dynamically debugs the programs shown in FIG. 1. In FIG. 6, for example, a debugger 40 is going to dynamically analyze the second program 20 that is under execution.

[0031]

In this case, when the first program 10 and the second program 20 start processing, the first and second decryption processing code modules 30 and 31 decrypt the first encrypted code module 11 to generate the first processing code module 1 which is executed by the first program 10.

[0032]

The second program 20, if debugged with the debugger 40, either temporarily stops execution or allows the user (operator) of the debugger 40 to execute steps, with a slowed-down processing speed.

[0033]

The first decryption processing code module 30 decrypts part of the encrypted code module 12 (resulting in incomplete processing code module 2) and, at the same time, the first decryption processing code module 30 decrypts part of the encrypted code module 13 (resulting in incomplete processing code module 3). When the first program 10 executes the incomplete processing code module 3, the operating system terminates the first program 10 assuming that an illegal processing has been performed.

[0034]

The second decryption processing code module 31 decrypts the incomplete processing code module 2 to generate the processing code module 2, and the second program 20 executes the processing code module 2. When the second decryption processing code module 31 decrypts part of the encrypted code module 14 (resulting in an incomplete processing code module 4), and the second program 20 executes the incomplete processing code module 4, the operating system terminates the second program 20 assuming that an illegal processing has been performed.

[0035]

If the processing of the second program 20, which is under debugging by the debugger 40, lags behind the processing speed shown in FIG. 3, the first program 10 that executes only partially decrypted processing code is terminated because an illegal processing has been performed. Because of this, the second program 20 that also executes only partially decrypted processing code is terminated because the illegal processing has been performed. At this time, the analysis by the debugger 40 also terminates.

[0036]

In this way, two programs -- first program 10 and second program 20 -- are structured so that they decrypt stepwise each other, i.e., one program decrypt one encrypted code module of the other program one by one. If either one of the programs is debugged, this program structure prevents the decryption procedure from being executed correctly (i.e., in a correct timing sequence) and terminates the programs because of improper processing. Therefore, the debugger 40 cannot complete a dynamic software analysis.

[0037]

Although the method described above uses two programs, that is, the first program 10 and the second program 20, it is also applicable not only to two programs but also to N programs (N is an integer equal to or larger than 3). When the number of the programs is increased to N, the first program 10 and the second program 20 decrypt encrypted code modules each other, ... program N-1 and program N decrypt encrypted code modules each other, and program N and the first program 10 decrypt encrypted code modules each other. This makes the programs more difficult to be analysed, as a whole.

[0038]

In FIG. 1 and FIG. 2, the decryption processing code modules 30 and 31 may be included in processing code modules 1 to 6. Including the decryption processing code modules 30 and 31 into any one of the processing code modules 1 to 6 in this manner further reduces the danger that a third party may analyze the programs.

[0039]

In addition, in the flowchart shown in FIG. 5, synchronization may be established not only by the waiting mechanism but also by a synchronized processing performed by a third program. In this case, this program sends a decryption permission message sequentially to the first program 10 and the second program 20. For example, upon recognising that step S11 has been ended, the third program sends the wait end message to the first program 10 waiting in step S2.

[0040]

In case where this method is employed, it is also possible to prevent the third program from dynamically analyzing the programs. To do so, an instruction coding is added to the waiting mechanism that causes control to be passed to the next step if no message is received for a predetermined period of time.

[0041]

FIG. 7 is a diagram showing the structure of programs in another embodiment of the present invention. Referring to FIG. 7, a first program 10 and a second program 20 each include a first decryption processing code module 30 and a second decryption processing code module 31, respectively. Encrypted code modules 11-13, which are decrypted each other, are placed, not in the first program 10 and the second program 20, but in a common program area 50 (that is provided separately).

[0042]

As shown in FIG. 7, the first and second programs 10 and 20 decrypt the encrypted code modules 11-13 in the common area 50 and execute the respective processing codes allocated to each of the first and second programs. In a system where rewriting (or transfer of) the encoded code module of another program is difficult because of the operating system specifications, this method gives same advantage as the one described above.

[0043]

The above disclosed method and process steps are realised by a computer readable program product comprising the steps for performing any one of the methods.

The meritorious effects of the present invention are summarized as follows.

According to the present invention described above, creating first and second programs that decrypt the corresponding processing code modules each other at execution time prevents a software debugger from making a dynamic software analysis.

It should be noted that other objects, features and aspects of the present invention will become apparent in the entire disclosure and that modifications may be done without departing the gist and scope of the present invention as disclosed herein and claimed as appended herewith.

Also it should be noted that any combination of the disclosed and/or claimed elements, matters and/or items may fall under the modifications aforementioned.